AD-A041 732 NAVAL RESEARCH LAB WASHINGTON D C F/G 9/2
TESTS OF RANDOM NUMBER GENERATORS FOR THE ASC.(U)
MAY 77 B T CARUTHERS, G H HERLING
UNCLASSIFIED NRL-MR-3509 NL

1 OF 1
ADA041732

END

DATE
FILMED
8 - 77

NRL Memorandum Report 3509

12 2

Tests of Random Number Generators for the ASC

B. T. CARUTHERS

Research Computation Center

G. H. HERLING

Radiation Technology Division

May 1977

NAVAL RESEARCH LABORATORY
Washington, D.C.

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: NRL Memorandum Report 3509
2. GOVT ACCESSION NO.
3. RECIPIENT'S CATALOG NUMBER

4. TITLE (and Subtitle): (6) Tests of Random Number Generators for the ASC
5. TYPE OF REPORT & PERIOD COVERED: (9) Interim report on a continuing NRL problem
6. PERFORMING ORG. REPORT NUMBER

7. AUTHOR(s): (10) B.T. Caruthers and G.H. Herling
8. CONTRACT OR GRANT NUMBER(s): (14) NRL-MR-3509

9. PERFORMING ORGANIZATION NAME AND ADDRESS
Naval Research Laboratory
Washington, D.C. 20375

10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
NRL General and Administrative Function 427316 and NRL Problem 66H01-48

11. CONTROLLING OFFICE NAME AND ADDRESS
Office of Naval Research, Arlington, VA 22217
and
Naval Air Systems Command (AIR-370) Wash., D.C. 20361

12. REPORT DATE: (11) May 1977
13. NUMBER OF PAGES: 34

14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office): (12) 34p.
15. SECURITY CLASS. (of this report): UNCLASSIFIED
15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Pseudo-Random Numbers
Random Number Generators
Vectorization

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

A scalar random number generator, RANDUL, and a vector random number generator, VRANF, are available on the ASC at NRL. Several tests to which they have been subjected are described, and the results indicate that both are generally acceptable for most applications.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-014-6601

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

251 950

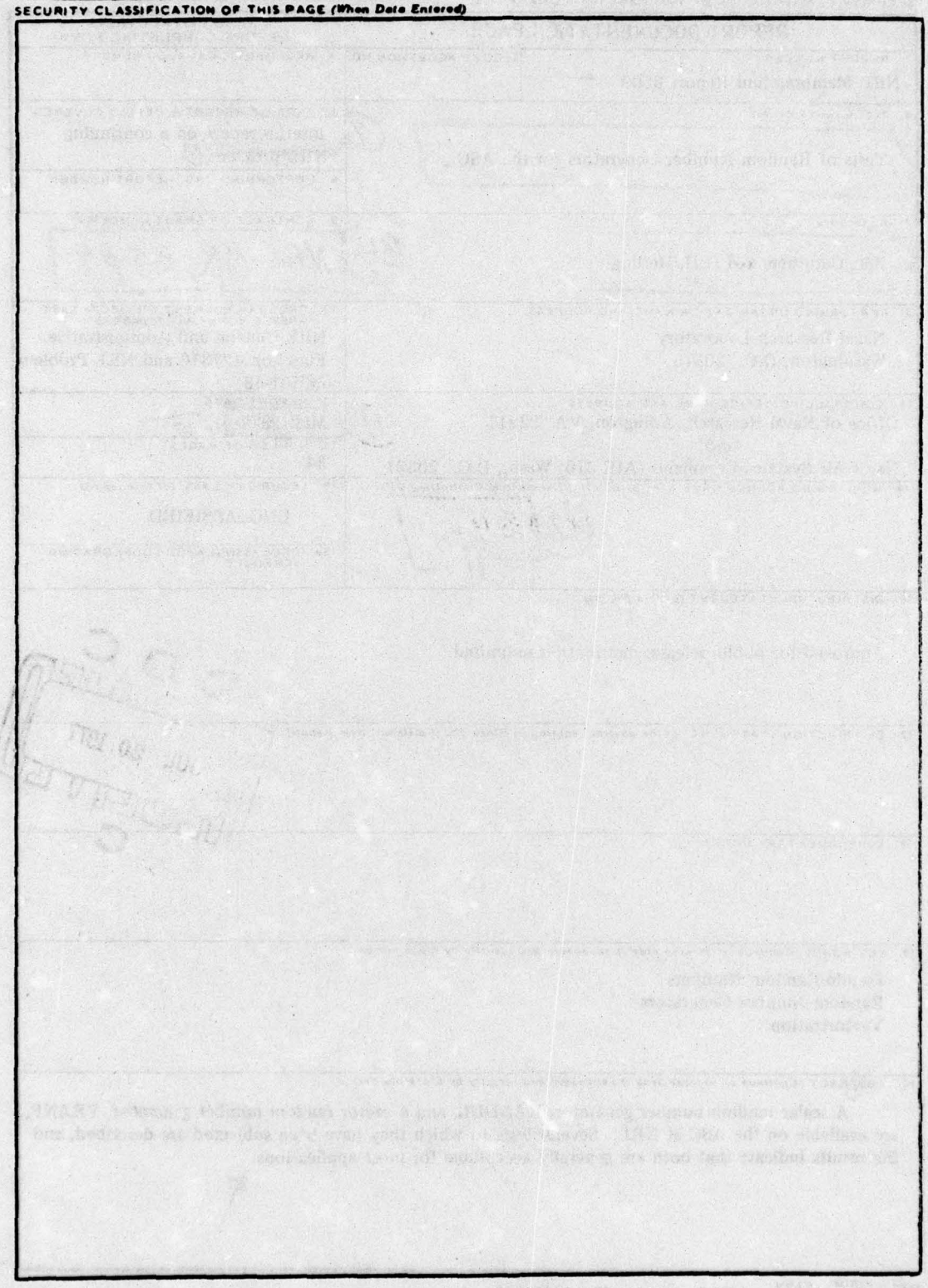

TABLE OF CONTENTS

TESTS OF RANDOM NUMBER GENERATORS FOR THE ASC

> Any one who considers arithmetical methods of producing random digits is, of course, in a state of sin.
>
> John von Neumann

> He that is without sin among you, let him first cast a stone ...
>
> John, 8:7

I. Introduction - Why so many tests

Because of its applications in simulation, statistical problems, and some areas of numerical analysis, an indispensable piece of software on a computer today is a copious, fast and reliable source of uniformly distributed random numbers. Early in computer history the idea of inputting a table of random numbers was rejected and the industry moved to a programmed arithmetic process for producing sequences of random numbers. If the definition of random means loosely "each number obtained by chance, having nothing to do with other numbers," then quite obviously numbers generated in such a deterministic way can only appear to be random and may be termed "pseudo-random".

The most popular and successful random number generators today use a linear congruential method which depends on the word size of the computer. A program RANDUL exists on the SPL and SYS.OLIB of the ASC

Note: Manuscript submitted April 29, 1977.

which implements this widely used industry algorithm. Knuth, however, warns of the pitfalls of blindly accepting handed down algorithms for random number generators[1] and Coveyou and MacPherson[2] warn of insufficiencies with using this algorithm and a 32-bit word size.

A second generator, VRANF, is also available on the SPL and SYS.OLIB of the ASC which uses a variation of the linear congruential algorithm and the computer's vector speed to generate its pseudo-random numbers. It is also a common fallacy that one can take a good generator, modify it a little and get an even more random sequence.[3]

These warnings raised the serious need for thorough testing of both algorithms. The theory of statistics provides us with quantitative measures of randomness; however, there is no end to the number of statistical tests that can be performed. Many tests were performed knowing that although not sufficient to prove randomness, every test passed built more and more confidence in the random sequence. Thus, the first set of questions the paper attempts to answer has to do with "goodness". (a) Do the two programs generate sequences which behave as if random or more simply are they both "good"? (b) If (a) is true which is better?

The second set of questions to be answered has to do with speed. (a) How fast are the routines? (b) Which is faster? Finally, is either good enough and fast enough to be highly recommended?

II. Algorithms

A. RANDUL

RANDUL uses the algorithm described by Lehmer[4] which generates residues of some integer by a linear transformation. In this method

one multiplies the current random integer I by a constant multiplier K and keeps the remainder after division by m:

$$\text{new I} = K \times \text{old I} \pmod{m}.$$

This is termed a linear congruential, or more specifically, multiplicative congruential method and is defined formally by the equation

$$x_{n+1} = ax_n \pmod{m}, \; n \geq 0 ,$$

where

x_o is the starting integer (seed), $x_o > 0$,

a is the integer multiplier, $a > 0$,

and

m is the modulus, $m > x_o$ and $m > a$.

The value of x_o should be odd, and the value of a should be large and odd. Another name for this method is the power residue method.

By choosing m to be the largest integer expressible in a single word plus one (for the ASC, $2^{31} + 1 = 2147483649$) calculation is fast and trivial. The residue calculation is automatically done by the computer's overflow handling of integer multiplication.

Some care must be taken in choosing a, the multiplier, to assure that the sequence has maximum period possible for a multiplicative generator. The maximum period, τ, for $m = 2^{32}$ is given by[5]

$$\tau(2^{\ell}) = 2^{\ell - 2} = 2^{30} = m/4 .$$

Maximum period, although desirable, is not sufficient for randomness. A proof given by Carmichael[6] and discussed by Knuth[5] shows that the

maximum period is achieved for 2^{ℓ} with $\ell > 3$ when x_o, the seed is odd, and $\underline{a} \equiv 3$ or $5 \pmod 8$. RANDUL uses the multiplier, $2^{16} + 3 = 65539$ which meets the condition $65539 \equiv 3 \pmod 8$.

The multiplier 65539 was chosen because it met the criterion mentioned above, appeared to be widely used, *i.e.*, found in much documentation,[7] and was used in the IBM-360's RANDU. Discussion of possible better choices will be covered in another section.

Once the power residue calculation takes place, the integer is saved as a seed for the (n + 1)-th call to the subroutine. The integer value is floated and multiplied by 2^{-31} to distribute the values on the unit interval.

B. VRANF

It is convenient to consider the algorithm of VRANF[8] to be composed of two sub-algorithms. The one initializes the routine, supplies the first 1000 pseudo-random numbers that are requested, and provides the starting point for fulfilling all later requests. The other satisfies all later requests. Both sub-algorithms employ integer arithmetic, and generate odd integers modulo 2^{28} which are then normalized in order to supply a finite set of allegedly uniformly distributed numbers chosen from the open interval (0.0,1.0).

The initialization algorithm is based upon the linear congruential sequence with the multiplier 1532455047, and the modulus 2^{32}. The sequence does not meet the criterion discussed earlier for maximum period, but only a small sample, the first 1000 numbers, is used.

I1. From an initial positive integer I_o, which is supplied either internally or by the user, calculate a seed x_1 which is large, positive, and odd.

I2. Generate

$$x_{n+1} = ax_n \pmod{2^{32}}, \quad 1 \le n \le 100,$$

and remove the three least significant bits.

I3. $$x_{n,1} = x_n \leftarrow 2x_n + 1, \quad 1 \le n \le 100,$$

and perturb the x_n in order to ensure that all are different but still odd.

I4. Generate

$$x_{n,k} = x_{k+n} = a^{k-1} x_{n,k-1} \pmod{2^{32}} \quad 2 \le k \le 10, \quad 1 \le n \le 100,$$

and remove the two least significant bits.

I5. Ensure that all $x_{n,k}$ are odd, unique and computed modulo 2^{28}.

At this point 1000 pseudo-random integers are available for normalization and transfer to the output array, and the first eight entries are stored for later use in step R3. It will be useful to denote these by x_{I+1000}, with $1 \le I \le 8$.

All remaining requests are generated according to the following algorithm:

R1. Choose a pseudo-random integer I

$$I = 7 \times 2^{-28} x_{677}, \quad 1 \le I \le 8.$$

R2. $x_n \leftarrow x_n x_{n+I} \pmod{2^{28}}, \quad 1 \le n \le 1000.$

R3. $x_{I+1000} \leftarrow x_I, \quad 1 \le I \le 8.$

R4. Normalize $x_n, \quad 1 \le n \le 1000,$ and transfer to output array.

To the authors' knowledge there is no rigorous proof that the algorithm given by R1-R3 produces uniformly distributed odd integers modulo 2^{28}. Rather, they appeal to the results of the statistical tests performed on the normalized, floating point results obtained from it.

III. Tests of Uniformity

Statistical tests were performed on RANDUL and VRANF for 10 seeds, the separate default seeds and nine randomly chosen seeds. A seed must be odd and no greater than 2147483647. They are listed in Table 1, and correspond to the seeds in the remaining tables.

TABLE 1. Seeds Used in Testing VRANF and RANDUL

Seed No.	Seed
1	123456781
2	379354235
3	246833
4	777654235
5	141597
6	73192881
7	429634011
8	2826305
9	1264396293
10[a]	392396625
10[b]	1009939513

[a]Default seed for RANDUL.

[b]Default integer for VRANF.

A. Uniformity of singles as measured by the χ^2-test

The most elementary test of equidistribution is to partition the unit interval into small subintervals of equal width and count the number of terms of any subsequences that fall in each subinterval.[9] The values are then compared with the expected values by means of a chi-square test.

The unit interval was divided into 32 cells using sample size 1000, 64 cells using sample size 5000, and 128 cells using sample size 10000.

Considering Tables 2 and 3, we can not say that the generated numbers are not uniform by singles since

$$\text{var}\,(\chi^2) = 2f,$$

where f is the number of degrees of freedom, and in any one column at most three values exceed $f + (2f)^{1/2}$, which is within acceptable limits.

TABLE 2. Uniformity Test Results - RANDUL

	1000 Nos. 32 groups		5000 Nos. 64 groups		10000 Nos. 128 groups	
Seed	χ_o^2	100 P ($\chi^2 \le \chi_o^2$)	χ_o^2	100 P ($\chi^2 \le \chi_o^2$)	χ_o^2	100 P ($\chi^2 \le \chi_o^2$)
1	30.7	51.6	64.4	57.2	127.7	53.5
2	41.1	89.4	85.6	96.9	154.4	95.0
3	32.1	58.9	68.0	68.8	102.4	5.3
4	37.7	81.0	55.8	27.2	98.2	2.7
5	26.6	30.9	77.7	90.0	156.3	96.0
6	41.0	89.3	70.0	74.6	129.8	58.5
7	20.2	6.8	51.9	16.1	127.6	53.1
8	32.0	58.3	56.6	29.7	159.7	97.4
9	30.8	52.6	71.7	78.8	127.5	52.8
10	32.3	59.7	59.5	39.9	125.7	48.5

TABLE 3. Uniformity Test Results - VRANF

	1000 Nos. 32 groups		5000 Nos. 64 groups		10000 Nos. 128 groups	
Seed	χ_o^2	100 P ($\chi^2 \le \chi_o^2$)	χ_o^2	100 P ($\chi^2 \le \chi_o^2$)	χ_o^2	100 P ($\chi^2 \le \chi_o^2$)
1	20.4	7.2	63.0	52.4	133.9	68.0
2	24.6	21.6	58.8	37.5	98.2	2.7
3	37.1	79.0	82.8	95.2	133.1	66.2
4	21.6	10.6	76.1	87.5	132.6	65.2
5	21.3	9.7	73.7	83.2	150.5	92.4
6	41.3	89.7	65.6	61.2	129.5	57.9
7	21.1	9.1	72.0	79.6	124.1	44.2
8	34.8	70.6	68.2	69.4	109.5	13.4
9	21.3	9.7	54.5	23.1	123.2	42.1
10	18.2	3.4	52.0	16.2	136.2	72.8

B. Uniformity of pairs as measured by the χ^2-test

Not only singles should be uniform on a line but also pairs should be uniform over a square.[9]

Successive pairs of pseudo-random numbers were taken as points in the unit square which was divided into 36 cells for 500 pairs, 64 cells for 2500 pairs, 100 cells for 5000 pairs. A χ^2-test was then performed on each of the distributions obtained, and the results are given in Tables 4 and 5.

The probabilities shown in the third and fifth columns of both tables suggest the possibility of dangerous geometrical correlations of successive pairs. On the other hand, the last column in each table is so impressive that we are inclined to believe that both generators are satisfactory with respect to the pair distribution.

Using the results to compare VRANF and RANDUL, we can not say that one significantly outperforms the other. In addition, we note that the results with the default seeds are at least as good if not better than any of the randomly chosen seeds.

TABLE 4. Pairwise Uniformity Test Results - RANDUL

	500 pairs 36 groups		2500 pairs 64 groups		5000 pairs 100 groups	
Seed	χ_o^2	100 P ($\chi^2 \leq \chi_o^2$)	χ_o^2	100 P ($\chi^2 \leq \chi_o^2$)	χ_o^2	100 P ($\chi^2 \leq \chi_o^2$)
1	25.7	12.7	69.6	73.4	98.5	50.5
2	44.5	86.9	85.9	97.1	112.6	83.4
3	24.2	8.4	51.2	14.5	94.7	39.7
4	38.6	68.8	56.6	29.9	95.1	40.7
5	33.2	44.0	83.7	95.8	91.9	32.0
6	58.0	99.1	72.7	81.2	101.3	58.2
7	30.1	29.5	61.0	45.3	123.7	95.3
8	49.5	94.7	60.5	43.3	76.7	4.7
9	35.2	54.4	51.8	15.9	109.4	77.7
10	21.4	3.5	47.9	7.9	83.4	13.1

TABLE 5. Pairwise Uniformity Test Results - VRANF

	500 pairs 36 groups		2500 pairs 64 groups		5000 pairs 100 groups	
Seed	χ_o^2	100 P ($\chi^2 \leq \chi_o^2$)	χ_o^2	100 P ($\chi^2 \leq \chi_o^2$)	χ_o^2	100 P ($\chi^2 \leq \chi_o^2$)
1	36.7	61.0	80.1	92.8	110.1	79.1
2	32.7	41.8	59.4	39.4	121.1	93.5
3	24.6	9.5	71.1	77.4	100.2	55.2
4	37.1	62.9	90.1	98.6	86.3	18.5
5	33.2	44.6	47.6	7.4	81.1	9.5
6	38.6	68.8	63.3	53.7	94.4	38.9
7	41.1	78.1	65.7	61.8	124.2	95.6
8	37.4	64.1	56.0	27.8	115.0	87.1
9	34.0	48.1	69.3	72.6	94.3	38.5
10	18.2	0.9	75.3	86.1	100.1	54.9

C. Uniformity of triplets as measured by the χ^2-test

Dividing the unit cube into equal cells and testing for uniformly distributed triplets affords another test of equidistribution. MacLaren and Marsaglia[9] reported a version of RANDUL on a 36 bit machine with the multiplier $2^{17} + 3$ does not produce a uniform density in three dimensions. Blood[10] explains algebraically how the triplets fall on a set of planes in the unit cube predetermined by the multiplier and the modulus and with RANDUL's multiplier, 66539, triplet correlations will always be found. MacLaren and Marsaglia's[9] test with the unit cube divided into 1000 equal cells and 3 triples taken from each set of 10 numbers, with a sample size of 30,000, was performed on VRANF. The results are given in Table 6.

TABLE 6. Triplet Uniformity Test Results - VRANF

Seed	χ_o^2	$100\,P(\chi^2 \leq \chi_o^2)$
1	998.9	49.4
2	1012	61.2
3	1112	99.3
4	1009	58.2
5	954.2	15.3
6	1067	93.2
7	1069	93.6
8	1062	91.6
9	1022	69.4
10	964.9	21.7

The probability, expressed as a percentage, that the appropriate chi-square variate will exceed the observed test value may be occasionally large or small, but the preponderance of large values leads to the conclusion that VRANF also fails the triplets test.

D. Uniformity as measured by the Kolmogorov-Smirnov goodness-of-fit test

Another more powerful method of testing equidistribution is the Kolmogorov-Smirnov (KS) goodness-of-fit test. The chi-square tests for distribution considers observations falling in a finite number of categories while the KS test looks at the random real number between 0 and 1 as a continuous distribution.

The statistics K_n^+ and K_n^- are calculated by

$$K_n^+ = \sqrt{n} \max_{-\infty < x < +\infty} (F_n(x) - F(x)) ,$$

and

$$K_n^- = \sqrt{n} \max_{-\infty < x < +\infty} (F(x) - F_n(x)) ,$$

where $F_n(x)$ is the observed distribution function and $F(x)$ is the expected distribution function. Knuth[11] remarks that using several tests for moderately sized n and then combining the observations later in another KS test will tend to detect both local and global nonrandom behavior. The test was performed with groups of 1000 observations 1000 times and a final KS test performed on these results, thus using a total sample size of 1,000,000.

TABLE 7. Application of Kolmogorov-Smirnov Goodness-of-Fit Test - RANDUL

Seed No.	No. of Local Failures	K_o [a]	100 P ($K \leq K_o$)
1	80	0.41	28.6
2	79	0.42	29.7
3	63	0.70	62.5
4	94	0.29	15.5
5	81	0.79	71.3
6	88	0.52	41.8
7	90	1.27	96.0
8	85	0.84	75.6
9	83	0.88	78.7
10	64	0.49	38.1

[a]Final KS statistic.

TABLE 8. Application of Kolmogorov-Smirnov Goodness-of-Fit Test - VRANF

Seed No.	No. of Local Failures	K_o [a]	100 P ($K \leq K_o$)
1	99	0.32	18.5
2	69	0.71	63.5
3	89	1.05	88.9
4	67	0.36	22.8
5	100	0.67	59.2
6	88	1.40	98.0
7	72	1.37	97.6
8	70	1.02	87.5
9	74	0.50	39.3
10	78	0.55	45.4

[a]Final KS statistic.

If we choose significance levels for the K_n^+ and K_n^- corresponding to probabilities greater than 95% or less than 5% and count the failures, we might expect a 10% failure rate. Tables 7 and 8 show the results of the 1000 KS tests with local failures falling within the expected values. However, the result of the KS test applied again to the 1000 individual test results detects a global nonrandomness using seeds 6 and 7 in VRANF and seed 7 in RANDUL. The KS test evinces RANDUL meets the criterion for randomness better than VRANF.

E. Runs test

A strong test recommended by Knuth,[12] strength being implied by the fact that a large number of random number generators fail it, is a test for "runs up" and/or "runs down". As implemented, the test tabulates "runs up" counting six categories of runs of lengths 1 through 5, and runs of length 6 or greater. Since adjacent runs are not independent the results are evaluated by the statistic

$$V = \frac{1}{n} \sum_{1 \le i,j \le 6} (\text{COUNT}(i) - nb_i)\,(\text{COUNT}(j) - nb_j)\,a_{ij}$$

where a_{ij} and b_i are tabular covariance values derived and calculated in Ref. (12). The statistic V then has a chi-squared distribution with 6 degrees of freedom. The results are presented in Tables 9 and 10.

RANDUL fails the runs test. Of the 50 entries in Table 9, sixteen exceed 0.9, which indicates a highly significant deviation from the hypothesis that the numbers are uniformly distributed. VRANF survives much better with only 8 of 50 entries exceeding 0.9 which we might term

TABLE 9. Probabilities that X^2 is Less Than the Observed Value for RANDUL Subjected to the Runs Test

Seed	Gp1	Gp2	Gp3	Gp4	Gp5
1	0.927	0.979	0.903	0.771	0.233
2	0.556	0.217	0.875	0.892	0.154
3	0.296	0.482	0.799	1.0	0.532
4	0.027	0.091	0.998	0.190	0.652
5	0.825	1.0	0.267	0.324	0.203
6	0.826	0.996	0.907	0.303	0.722
7	0.781	0.743	0.782	0.820	1.0
8	0.996	0.409	0.875	1.0	0.967
9	0.569	0.995	0.780	0.993	0.925
10[a]	0.095	0.115	0.095	0.916	0.277

[a]The default sequence for which the group size is 4133 numbers. For all other seeds the group size is 5133 numbers.

TABLE 10. Probabilities that X^2 is Less Than the Observed Value for VRANF Subjected to the Runs Test

Seed	Gp1	Gp2	Gp3	Gp4	Gp5
1	0.698	0.034	0.969	0.071	0.347
2	0.355	0.536	0.083	0.489	0.917
3	0.216	0.458	0.260	0.111	0.504
4	0.557	0.773	0.418	0.985	0.120
5	0.352	0.565	0.505	0.154	0.489
6	0.622	0.728	0.942	0.747	0.761
7	0.941	0.349	0.132	0.851	0.296
8	0.444	0.946	0.701	0.518	0.981
9	0.664	0.457	0.800	0.317	0.566
10[a]	0.862	0.226	0.998	0.224	0.588

[a]The default sequence for which the group size is 4133 numbers. For all other seeds the group size is 5133 numbers.

as not showing a significant deviation from the hypothesis that the numbers are uniformly distributed.

F. Difference test

In the absence of a rigorous proof, the author of VRANF elicits its validity by its performance on a test based upon differences.[8]

If we have two statistically independent random variables distributed uniformly, x_1 and x_2, the difference variable $y_1 = x_2 - x_1$ has a triangular distribution, and the variable

$$z_1 = y_1 + 1 \qquad y_1 \leq 0$$

$$= y_1 \qquad y_1 > 0$$

again has a uniform distribution. Using this principle, we may test the outputs of a random number generator, $x_1, x_2, \ldots, x_N, x_{N+1} = x_1$, their first differences, $z_1, z_2, \ldots, z_N, z_{N+1} = z_1$, the first differences of the z-variables, etc. for uniformity.

A sample of 10000 numbers was chosen to start, the first 50 differences of the x-variables were calculated, and uniformity was tested by means of the χ^2-test with the unit interval divided into 25 cells.

Figure 1 shows plots of the theoretical χ^2-distribution for 24 degrees of freedom, and the observed results obtained from the default sequences of RANDUL and VRANF. It illustrates that VRANF remains stable for 50 differences, and the break in the curve for RANDUL at a percentile value of about 40 illustrates that its randomness collapses at the twenty-second difference. VRANF consistently, for all seeds, matched the expected chi-square distribution for 24 degrees of freedom while RANDUL failed at the twenty-second difference for all seeds.

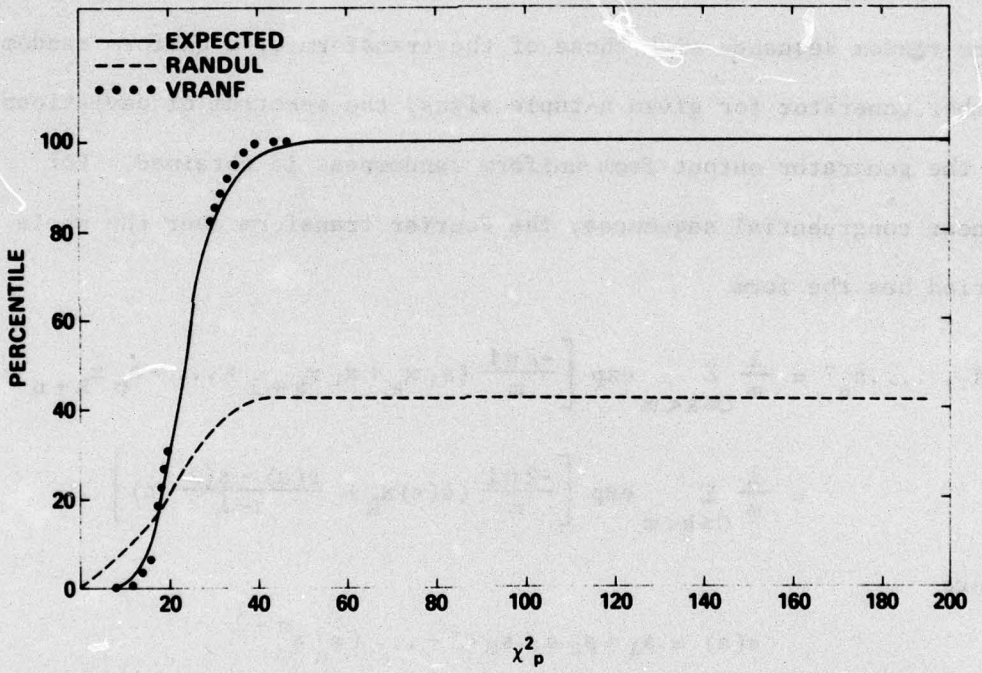

Fig. 1 — Chi-square distribution of differencing test for default seeds for 24 degrees of freedom. Results are shown for theoretical distribution and the observed results obtained from RANDUL and VRANF.

G. Spectral test

Coveyou and MacPherson[2] formulated a test to measure the statistical independence of successive n-tuples of random numbers using finite Fourier analysis. In a proof detailed in their paper, it is shown that the Fourier coefficients are sufficient statistics themselves, and a computational method is presented for calculating the coefficients over the whole period for a linear congruential generator. By comparing the Fourier coefficients of the transform of a truly uniform random sequence with those of the transform of a uniform random number generator for given n-tuple sizes, the spectrum of deviations of the generator output from uniform randomness is obtained. For linear congruential sequences, the Fourier transform over the whole period has the form

$$f(s_1, \ldots, s_n) = \frac{1}{m} \sum_{0 \le k < m} \exp\left[\frac{-2\pi i}{m}(s_1 x_k + s_2 x_{k+1} + \ldots + s_n x_{k+n-1})\right]$$

$$= \frac{1}{m} \sum_{0 \le k < m} \exp\left[\frac{-2\pi i}{m}(s(a) x_k + \frac{s(a) - s(1)}{a-1} c)\right],$$

where

$$s(a) = s_1 + s_2 a + s_3 a^2 + \ldots + s_n a^{n-1},$$

and $\underline{a}$ is the multiplier. Assuming the sequence has maximum period this reduces to the sum of a geometric series

$$\frac{1}{m} \sum_{0 \le k < m} \exp\left[\frac{-2\pi i}{m}(s(a)k + \frac{s(a) - s(1)}{a-1} c)\right] \delta\left(\frac{s(a)}{m}\right),$$

where $\delta(x) = 1$ if x is an integer, and 0 otherwise. The quantity $f(s_1, \ldots, s_n)$, therefore, either vanishes or has unit magnitude.

We may physically interpret $f(s_1, \ldots, s_n)/m^n$ as the amplitude of the n-dimensional plane wave

$$\omega(t_1, \ldots, t_n) = \exp\left[2\pi i \left(\frac{s_1}{m} t_1 + \ldots + \frac{s_n}{m} t_n\right)\right] ,$$

and assign a wave number ν corresponding to its frequency

$$\nu = \sqrt{s_1^2 + \ldots + s_n^2} \quad \text{when } |s_k| \leq \frac{m}{2} .$$

For linear congruential sequences of maximum period, the smallest nonzero wave number in the spectrum is given by

$$\nu_n = \min \sqrt{s_1^2 + s_2^2 + \ldots + s_n^2} ,$$

where the minimum is taken over all n-tuples of integers excluding $(s_1, s_2, \ldots, s_n) = (0, 0, \ldots, 0)$ subject to the constraint $s(a) \equiv 0 \pmod m$. To rate the effectiveness of a multiplier in a linear congruential sequence of maximum period we calculate

$$C_n = \frac{\pi^{n/2} \nu_n^n}{(n/2)!\, m} .$$

Values of C_n of order unity or larger correspond to randomness, small values correspond to nonrandomness. Large values imply large wave number (short wavelength) deviations that will tend to average out over the entire cycle. Long wavelengths imply a systematic departure from uniformity.

Implementing the spectral test involved using the ASC extended precision package for integer arithmetic, SAC-1, since the computer algorithm described in Ref. (13) requires exact integer operations.

The author of Ref. (13) qualifies the algorithm by stating, "the method will get into an infinite loop in some cases." Modifying the algorithm to avoid infinite loops leads to a very long search for the quantity ν_n. Since the extended precision package is implemented by means of a linked list, it is very space and time consuming, and a lengthy search may be prohibitive. This was the case in solving for coefficients C_3, C_4, and C_5 for VRANF as indicated in Table 11.

TABLE 11. Spectral Test Results

Coefficients	RANDUL	VRANF
C_1	2.0	2.0
C_2	3.1412	2.19
C_3	0.0000025	*
C_4	0.00003092	*
C_5	0.000355	*

*Non-converging algorithm.

From results shown in Table 11 RANDUL must be rejected as a generator where the uniformity of distribution of n-tuples greater than 2 is a critical factor.

The statistics shown for VRANF are not as potent since the spectral test rates, over the full period, the multiplier of a linear-congruentially generated sequence, and the congruential algorithm is only used to generate the first 1000 numbers in VRANF. A true rating of VRANF by a theoretical test is non-trivial since the full period has not been mathematically predicted.

F. A. Blood[10] discusses and predicts the harmful correlations found by the spectral test in the power residues generated by RANDUL. He shows algebraically how to pick a multiplier to eliminate the problem of geometrical correlations of n-tuples 3 or greater and supplies a set of multipliers which meet his criterion.

H. Autocorrelation test

A quantity of interest prior to employing pseudo-random numbers in many applications is the autocorrelation function of such a sequence. Several definitions of this quantity exist, and for the present purpose it is convenient to choose one that bounds it by unit magnitude. The serial correlation coefficients provide such a definition and are given by

$$C(\ell) = \frac{E[(x_n - \bar{x})(x_{n+\ell} - \bar{x})]}{E[(x_n - \bar{x})^2]},$$

where

$$E[Z_n] = (1/N) \sum_{n=1}^{N} Z_n,$$

and $\bar{x} = E[x_n]$, with the x_n being the pseudo-random numbers that have been generated by the routine under test. The convolution appearing in the numerator has been calculated by first obtaining its finite Fourier transform with the routine FOURTR,[14] which reduces the convolution to multiplication, and then inverting the result. With this technique, the values of $C(\ell)$ have been obtained from the first 2^{14} numbers generated by RANDUL and VRANF for the seeds given in Table 1,

and for $0 \le \ell \le 2^{14}$. The sample means and the values of $C(\ell)$ for $\ell = 1, 2, 4, 8, 16$ are given in Tables 12 and 13 for RANDUL and VRANF respectively.

Various assumptions, all of which involve the normal distribution at some stage, about the quantities for which the serial correlations are of interest, indicate that for $\ell > 0$ and large n, the $C(\ell)$ tend to be normally distributed with mean and variance $\mathcal{O}(1/n)$.[15] With the assumption that this theory is applicable to the serial correlation coefficients of a sequence obtained from the uniform distribution, the probability of obtaining $|C(\ell)|$ greater than $1.6\,n^{-1/2} = 0.0125$ is 10%, and that of obtaining a result greater than $2\,n^{-1/2} = 0.0156$ is 5%.

Of the fifty entries in the last five columns of Table 12, there are three greater than the 5% critical point, and four greater than the 10% critical point. Similarly in Table 13, the corresponding numbers are two and four. These results indicate that both RANDUL and VRANF have acceptable serial correlation properties.

TABLE 12. Results of the Autocorrelation Test Applied to RANDUL and Sample Mean Values

Seed No.	$\bar{x}$	C(1)	C(2)	C(4)	C(8)	C(16)
1	0.4984	-0.0102	0.0020	0.0099	0.0170	0.0200
2	0.5042	-0.0009	0.0043	0.0031	0.0001	-0.0066
3	0.5027	-0.0083	-0.0031	0.0052	-0.0159	0.0060
4	0.5009	0.0053	-0.0121	-0.0057	-0.0136	0.0086
5	0.5003	-0.0037	0.0083	0.0006	-0.0080	-0.0100
6	0.5035	-0.0027	-0.0017	0.0022	0.0018	0.0018
7	0.4984	-0.0081	0.0074	-0.0102	-0.0057	-0.0060
8	0.5005	-0.0030	-0.0107	-0.0088	0.0020	-0.0121
9	0.5026	0.0012	-0.0043	-0.0104	0.0014	0.0000
10	0.4997	0.0116	0.0019	-0.0089	-0.0024	0.0007

TABLE 13. Results of the Autocorrelation Test Applied to VRANF and Sample Mean Values

Seed No.	$\bar{x}$	C(1)	C(2)	C(4)	C(8)	C(16)
1	0.4932	-0.0027	-0.0207	-0.0009	0.0105	0.0069
2	0.4996	-0.0040	0.0000	0.0069	-0.0184	-0.0012
3	0.5006	0.0003	-0.0036	0.0115	0.0085	0.0117
4	0.5009	0.0056	-0.0070	-0.0092	-0.0086	-0.0032
5	0.5044	0.0013	-0.0113	-0.0029	-0.0090	-0.0030
6	0.4982	-0.0137	0.0052	-0.0129	0.0083	-0.0020
7	0.5011	-0.0027	0.0092	-0.0001	-0.0012	-0.0056
8	0.4986	-0.0028	-0.0070	-0.0124	-0.0120	0.0115
9	0.4991	0.0079	0.0001	-0.0005	-0.0002	0.0005
10	0.5019	0.0045	-0.0140	-0.0100	0.0049	0.0163

IV. Timing Tests

The subroutine RANDUL must be called once for every number generated. The routine VRANF, however, uses the machine vector capability to generate and store an array of 1000 numbers regardless of the value N, the number of numbers requested. The N numbers are then moved to the output array. Subsequent calls generate a new array of 1000 numbers when the original 1000 are depleted. Thus, time-wise use of VRANF is optimized when N = 1000.

Results of tests in Table 14 show VRANF is faster than RANDUL by a factor of 1000 when the former is used at its maximum. However, RANDUL is a more efficient generator when only a few numbers are desired. Timing was performed by routine SECOND.

TABLE 14. Timing Tests RANDUL vs. VRANF

Test No.	Test Description	RANDUL time in sec.	VRANF time in sec.
1	Default seed - Single call N = 1	0.31×10^{-4}	0.95×10^{-3}
2	Seed initialization only	0.26×10^{-4}	0.93×10^{-3}
3	Average time - 1000 calls N = 1	0.32×10^{-4}	0.44×10^{-4}
4	N = 1000[a]	0.20×10^{-1}	0.21×10^{-4}
5	VRANF - Single call N = 1001		0.41×10^{-4}
6	VRANF - Single call N = 3000		0.63×10^{-4}

[a] Includes program looping for RANDUL.
Single call to VRANF with N = 1000.

V. Recommendations and Restrictions

We conclude that VRANF outperforms RANDUL both in speed, period, and somewhat in statistical performance, although both have difficulty uniformly filling the unit cube.

RANDUL is recommended for usage when: 1. generating a small number of numbers; 2. the user desires the property that an individual number will not repeat until the whole sequence repeats; 3. the user desires random integers of range 1 to $2^{31} + 1$.

VRANF is recommended for general usage, however, the user might note a few potential problems discovered in using it. The seed value supplied in the subroutine call is altered by VRANF and the altered value returned. Calling VRANF without a call to SETVR for initializing a seed presently does not generate the same sequence as calling SETVR(0). VRANF documentation, however, purports this to be true. The routine will be changed to generate duplicate sequences in this case, in order to conform with existing documentation. To use the subroutine efficiently the user is responsible for dispersing the supplied array of random numbers, and requesting more when the queue is empty, which may involve changing the logic of existing routines currently using a random number.

Neither routine can be used for producing matching sequences on the ASC and CDC. There exists a slow random number generator, RANF, for this purpose which is recommended for use only for the duplicate sequence problem.[16]

VI. Acknowledgments

The authors wish to thank Mr. Paul Anderson for helpful discussions about extended precision integer arithmetic on the ASC, and Dr. Richard Roth for instructive comments about VRANF. They are also grateful to Mrs. Laura C. Davis for supplying a CDC version of the spectral test.

References

1. Knuth, D. E., The Art of Computer Programming; Seminumerical Algorithms, vol. 2, Addison-Wesley, Reading, MA (1969) 4.

2. Coveyou, R. R., and MacPherson, R. D., Fourier analysis of uniform random number generators, Jour. Assoc. Comp. Mach. 14 (1967) 100.

3. Knuth, D. E., op. cit., 25.

4. Lehmer, D. H., Mathematical methods in large-scale computing units, Proc. Second Symp. on Large-Scale Digital Computing Machinery, Harvard University Press, Cambridge, MA (1951) 141.

5. Knuth, D. E., op. cit., 17.

6. Carmichael, R. D., Note on a new number theory function, Bull. Amer. Math. Soc. 16 (1910) 232.

7. Schwartz, M., and Shaw, L., Signal Processing: Discrete Spectral Analysis, Detection, and Estimation, McGraw-Hill, New York, NY (1975) 103.

8. Roth, R., unpublished.

9. MacLaren, M. D., and Marsaglia, G., Uniform random number generators, Jour. Assoc. Comp. Mach. 12 (1965) 83.

10. Blood, F. A., Jr., Correlations in power residue generated random numbers, Jour. Comp. Phys. 20 (1976) 372.

11. Knuth, D. E., op. cit., 41.

12. Knuth, D. E., op. cit., 60.

13. Knuth, D. E., op. cit., 82.

14. ASC Mathematical Library, Texas Instruments, Inc., Pub. No. 929978-2, March (1975) 301.

15. Dixon, W. J., Further contributions to the problem of serial correlation, Ann. Math. Stat. 15 (1944) 19.

16. R. McGill, private communication.

Appendix A. RANDUL - Availability and Source Listing

A. RANDUL resides on the SPL whose objects are currently on SYS.OLIB.

```
      SUBROUTINE RANDUL(IY,YFL)
CD   1. IDENTIFICATION NAME:    2. CLASSIFICATION:    3. DATE:
CD      RANDUL                      G5                    090476
CD   4. TITLE:
CD      UNIFORMLY DISTRIBUTED RANDOM NUMBER GENERATOR
CD   5. AUTHOR(S); ORGANIZATION(S):
CD      UNKNOWN ; OAKRIDGE LABORATORY
CD      MODIFIER(S); ORGANIZATION(S):
CD      TERRY CARUTHERS; NRL RCC:  G.H. HERLING; NRL RTD
CD      CONVERTER(S); ORGANIZATION(S):
CD      TERRY CARUTHERS; NRL RCC
CD      REVIEWER; ORGANIZATION:
CD      SIDNEY PRAHL; TEXAS INSTRUMENTS, INC.
CD   6. DESCRIPTION/PURPOSE (SPECIFY SUBROUTINE, FUNCTION, OR PROGRAM):
CD      SUBROUTINE GENERATES A UNIFORMLY DISTRIBUTED PSEUDO-RANDOM
CD      NUMBER ON THE INTERVAL(0.0,1.0), END POINTS EXCLUDED, AND/OR
CD      INTEGER ON THE INTERVAL(0,2**31-1), USING LINEAR CONGRUENTIAL
CD      ALGORITHM.  THE USER MAY SUPPLY A STARTING SEED BY A SINGLE CALL
CD      TO RANDUS(ISEED) BEFORE CALLING RANDUL, OR CALL RANDUL ALONE
CD      WHICH SUPPLIES A DEFAULT SEED.  SUCCESSIVE CALLS TO RANDUL CAN
CD      GENERATE A NONREPEATING SEQUENCE OF MAXIMUM LENGTH, 2**29.
CD      BASIC USE IS FOR QUICK GENERATION OF SHORT SEQUENCES OF PSEUDO-
CD      RANDOM NUMBERS THAT APPEAR TO HAVE BEEN RANDOMLY SELECTED FROM
CD      THE UNIFORM PROBABILITY DENSITY.
CD   7. CALLING SEQUENCE OR OPERATIONAL PROCEDURE:
CD      CALL RANDUL(IY,YFL)
CD   8. ARGUMENTS (TYPE AND SIGNIFICANCE) AND/OR INITIAL CONDITIONS:
CD      IY-INTEGER,OUTPUT RANDOM INTEGER
CD      YFL-FLOATING PT, OUTPUT RANDOM NUMBER
CD      ISEED-INPUT INTEGER, 1 TO 10 DIGIT ODD INTEGER .LE. 2**31-1
CD            (2147483647) WHICH DETERMINES THE UNIQUE SEQUENCE.
CD      NOTE:YFL=NORMALIZED IY.
CD   9. ENTRY POINTS (WITH ARGUMENTS):
CD      CALL RANDUL(IY,YFL)
CD      CALL RANDUS(ISEED)
CD  10. LANGUAGE/LIMITATIONS/ACCURACY/SYSTEM DEPENDENCES:
CD      RANDUL IS A MULTIPLE ENTRY SINGLE PRECISION FORTRAN IV SUBROUTINE
CD      AND MUST BE COMPILED UNDER THE C OPTION OF THE NX COMPILER AS THE
CD      ALGORITHM DEPENDS UPON OVERFLOW AND UNDERFLOW TO SURVIVE.  THIS
CD      SUBROUTINE IS TRANSFERABLE ONLY TO ANOTHER 32 BIT MACHINE WHICH
CD      ALLOWS OVERFLOW/UNDERFLOW AND RETAINS THE LEAST SIGNIFICANT PART
CD      OF THE INTEGER MULTIPLY OVERFLOW.  THE SUBROUTINE ALSO NEEDS A
CD      CONSISTENT ENTRY POINT STRUCTURE.
CD  11. ROUTINES CALLED:
CD      NONE
      DATA IX/392396625/
      IY=IX*65539
      IF(IY) 5,6,6
5     IY=IY+2147483647+1
6     YFL=IY
      IX=IY
      YFL=YFL*.4656613E-9
      RETURN
      ENTRY RANDUS(IX)
      RETURN
      END
```

Appendix B. VRANF - Availability and Source Listing

A. VRANF resides on SYS.OLIB.

```
      SUBROUTINE VRANF (RA,LEN)
CD  1. IDENTIFICATION NAME:   2. CLASSIFICATION:   3. DATE:
CD     VRANF                        G5                 08/76
CD  4. TITLE:
CD     UNIFORMLY DISTRIBUTED VECTOR RANDOM NUMBER GENERATOR
CD  5. AUTHOR(S); ORGANIZATION(S):
CD     DICK ROTH-TI AUSTIN
CD     MODIFIER(S); ORGANIZATION(S):
CD     NONE
CD     CONVERTER(S); ORGANIZATION(S):
CD     TERRY CARUTHERS RCC-NRL  G.H. HERLING,RTD-NRL
CD     REVIEWER; ORGANIZATION:
CD     SIDNEY PRAHL: TEXAS INSTRUMENTS, INC.
CD  6. DESCRIPTION/PURPOSE (SPECIFY SUBROUTINE, FUNCTION, OR PROGRAM):
CD     THIS ROUTINE USES VECTOR SPEED TO GENERATE AN ARRAY OF RANDOM
CD     NUMBERS UNIFORMLY DISTRIBUTED IN THE RANGE (0.0,1.0),END POINTS
CD     EXCLUDED. AN INTRICATE ALGORITHM COMBINING LINEAR CONGRUENTIAL
CD     GENERATION WITH RANDOM CROSS MULTIPLICATION IS USED. USER MAY
CD     SUPPLY A STARTING SEED BY A SINGLE CALL TO SETVR(INIT) BEFORE
CD     CALLING VRANF,OTHERWISE A BUILT IN VALUE IS USED. AFTER ANY CALL
CD     VRANF MAINTAINS 1000 PARALLEL STREAMS OF RANDOM NUMBERS AND
CD     REGENERATES ONLY WHEN THESE ARE USED
CD  7. CALLING SEQUENCE OR OPERATIONAL PROCEDURE:
CD     CALL VRANF(RA,LEN)-STANDARD SUBROUTINE ENTRY
CD     CALL SETVR(INIT)-ENTRY POINT FOR INITIAL SEED MODIFICATION.
CD  8. ARGUMENTS (TYPE AND SIGNIFICANCE) AND/OR INITIAL CONDITIONS:
CD     RA-REAL-STARTING ADDRESS OF OUTPUT RANDOM ARRAY
CD     LEN-INTEGER-LENGTH OF ARRAY TO BE FILLED
CD     INIT-INTEGER- 1 TO 10 DIGIT ODD INTEGER .LE. 2**31-1(2147483647)
CD                   STARTING SEED WHICH DETERMINES UNIQUE SEQUENCE.
CD  9. ENTRY POINTS (WITH ARGUMENTS):
CD     CALL SETVR(INIT)-USED AS A ONE TIME ONLY CALL TO GENERATE A
CD                      UNIQUE SEQUENCE OF RANDOM NUMBERS. ZERO AS A
CD                      SEED GENERATES THE DEFAULT SERIES. WITH NO CALL
CD                      TO SETVR VRANF SUPPLIES A SEED,1482123447, WHICH
CD                      ALWAYS PRODUCES THE SAME SEQUENCE OF PERIOD
CD     2**27000.
CD     ARGUMENT-INIT -1-10 DIGIT ODD INTEGER
CD 10. LANGUAGE/LIMITATIONS/ACCURACY/SYSTEM DEPENDENCES:
CD     ROUTINE IS A MULTIPLE ENTRY SINGLE PRECISION FORTRAN IV SUB-
CD     ROUTINE AND MUST BE COMPILED UNDER O OPTION OF NX COMPILER, AS
CD     THE ALGORITHM DEPENDS UPON OVERFLOW AND UNDERFLOW TO SURVIVE.
CD     COMPILING UNDER K OPTION PRODUCES A VECTOR HAZZARD THAT MAY BE
CD     SAFELY IGNORED UNDER L OPTION. ROUTINE IS TRANSFERABLE ONLY TO
CD     ANOTHER 32 BIT MACHINE WHICH ALLOWS OVERFLOW/UNDERFLOW AND
CD     INTEGER MULTIPLY OVERFLOW. ALSO NEEDS CONSTSTENT ENTRY POINT
CD     STRUCTURE. ROUTINE IS WRITTEN WITH TI VECTOR PHILOSOPHY IN MIND.
CD     ADVANTAGES OF SPEED WILL BE LOST IF EXECUTED AS SCALAR CODE
CD 11. ROUTINES CALLED:
CD     NONE
      DIMENSION JR(1008),JRE(100,10),RA(LEN)
      EQUIVALENCE (JR,JRE)
      DATA  ISTART,  ISEED,  JAND,  KNORM,JSET,JREM,TNORM
     * / Z3C327439,Z5B5768B7,Z7FFFFFFF,Z3A100000,  0,1000,  0.0  /
C
C  RETURN WITH NO ACTION IF LEN < 1.
      IF (LEN .LT. 1) RETURN
```

```
C
C  INITIALIZE FIRST 1000 NOS. IF THIS HAS NOT BEEN DONE.
      IF (JSET .EQ.0) GO TO 10
   80 NEED = LEN
   90 NEAD = NEED
      KREM = JREM

C
C  DO I HAVE ENOUGH UNUSED NUMBERS TO FILL THIS REQUEST?
      IF (JREM .GT. NEED) GO TO 100
C
C  NO. BUT I'LL USE WHAT I HAVE AVAILABLE FOR NOW.
      NUM = JREM

      NEED = NEED - JREM
      JREM = 0
      GO TO 110
C
C  YES, I HAVE ENOUGH.
  100 NUM = NEED
      JREM = JREM - NEED
      NEED = 0
C
C  FLOAT NUMBERS AND MOVE TO OUTPUT ARRAY.
  110 DO 120 N = 1,NUM
  120 RA(N+LEN-NEAD) = JR(N+1000-KREM)
C
C  DID I MOVE ALL THAT WERE REQUESTED?
      IF (NEED .GT. 0) GO TO 140
C
C  YES.  REQUEST IS FILLED.  NOW NORMALIZE TO RANGE (0.0,1.0) BY
C  MULTIPLICATION BY RNORM.
      DO 130 N = 1,LEN
  130 RA(N) = RNORM*RA(N)
      RETURN
C
C  NO.  I DIDN'T HAVE ENOUGH.  SO NOW I GENERATE NEXT 1000 NUMBERS.
  140 JP = JR(677)*INORM + 1
      DO 150 J = 1,1000
  150 JR(J) = AND (JR(J)*JR(J+JP),JAND)
      DO 160 J = 1,8
  160 JR(J+1000) = JR(J)
      JREM = 1000
C
C  NOW GO BACK AND COMPLETE REST OF REQUEST.
      GO TO 90
C
C  JUMP TO HERE IF VRNDM CALLED WITHOUT PRIOR CALL TO SETVR.
10    IFLG = 1
      INT = ISTART
      GO TO 20
C
C  **********
C
```

```
      ENTRY SETVR (INIT)
      INT = (INIT+756831)*(INIT+9345872) + ISTART
      IFLG = 0
   20 JSET = 1
      JREM = 1000
      TNORM = 7*KNORM
      JR(1) = 2*INT*ISTART + 1
C
C  GENERATE FIRST 100 RANDOM NUMBERS IN SCALAR LOOP.
      J = 2
   30 JR(J) = LSHF(ISEED*JR(J-1),-3)
      J = J + 1
      IF (J.LT.101) GO TO 30
      DO 32 J = 1,100
   32 JR(J) = 2*JR(J) + 1
C
C  MODIFY NUMBERS TO AVOID MULTIPLE DUPLICATE SEQUENCES
      DO 35 J = 1,100
   35 JR(J) = JR(J) + 15486362*J
C
C  USE VECTORIZED LOOPS TO EXTEND SEED TO LENGTH OF 1000
      DO 55 K = 2,10
      IF (JREM.LT.J) GO TO 55
      DO 50 J = 1,100
   50 JR(J,K) = LSHF(ISEED*JR(J,K-1),-2)
   55 CONTINUE
      DO 57 J = 1,1000
   57 JR(J) = OR(JR(J),1)
C
C  MODIFY NUMBERS TO AVOID MULTIPLE DUPLICATE SEQUENCES
      DO 60 J = 1,1000
   60 JR(J) = JR(J) + 1768534*J
      DO 70 J = 1,1000
   70 JR(J) = AND(JR(J),JAND )
      DO 72 J = 1,8
   72 JR(J+1000) = JR(J)
C
      IF (IFLG.EQ.0) RETURN
      GO TO 80
C
      END
```